Параничев А.В., аспирант
Поршнев С.В., проф., д-р техн. наук

ВИЗУАЛЬНОЕ РАСПОЗНАВАНИЕ ЭЛЕМЕНТОВ ГРАФА

Структурное описание элементов разных объектов широко применяется при моделировании информационных и технических систем. В основе такого подхода к распознаванию образов лежит аппарат для описания информативных особенностей структур в виде их метрических и топологических свойств. При этом могут быть решены задачи обнаружения закономерных связей между структурными свойствами, поиска необходимых структурных фрагментов для последующего оценивания параметров системы.

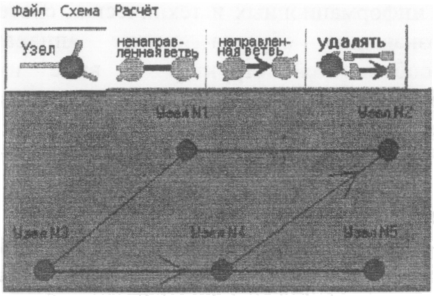
Иерархия распознаваемой структуры предполагает описание сложных объектов с помощью более простых подобъектов, что в общем случае может быть представлено в виде элементов (узлов и ветвей) графа. В этом случае для описания системы следует использовать парадигму объектно-ориентированного программирования, разбив при этом все имеющиеся в распоряжении данные на однородные классы, и решать далее поставленную задачу отдельно для каждого такого класса. В соответствии с выбранным подходом следует описать базовые классы узла и ветви как элементов, образующих распознаваемую структуру, поскольку любой объект можно представить в виде модели узла или ветви графа.

Применим выбранный подход к двухполюсным структурам. В этом случае распознаваемый образ представляет собой узел или ветвь и отображается графически в зависимости от характеристик конкретного объекта. При этом целесообразно устанавливать факт распознавания конкретного образа, учитывая только общие характеристики всех рассматриваемых узлов и ветвей.

Проиллюстрируем вышесказанное на примере создания и редактирования двухполюсных структур с помощью широко используемой операции графического интерфейса «Drag–And–Drop» («перетащить и оставить»). Для определенности, в качестве среды реализации данной задачи выберем пакет C++ Builder 6. Так, на рисунке представлен фрагмент интерфейса программы, работающей под управлением Windows, для реализации техники «Drag–And–Drop». В этом случае в качестве холста для графических объектов использован компонент TPaintBox, в области которого отслеживаются нажатие, последующее перемещение и отпускание левой клавиши мыши. При этом вызываются, соответственно, события *PaintBox1MouseDown* (/*другие параметры*/, int *X*, int *Y*), *PaintBox1MouseMove* (/*другие параметры*/, int *X*, int *Y*) и *PaintBox1MouseUp* (/*другие параметры*/, int *X*, int *Y*), принимающие в качестве входных параметров текущие координаты мыши *X* и *Y*.

Интерфейс позволяет вводить узлы и ветви (как направленные, так и ненаправленные), а также удалять узлы с инцидентными им ветвями посредством клавиш быстрого доступа, которые являются экземплярами

компонента TSpeedButton. Здесь следует отметить, что в каждый момент времени только одна из четырех клавиш быстрого доступа нажата (рис. 1). При этом глобальная переменная nEditType устанавливается в соответствующее значение. Это дает возможность распознавания нажатой клавиши быстрого доступа при щелчке левой клавиши мыши в области компонента TPaintBox1.



Фрагмент работы программы, реализующей технику «Drag-And-Drop»

Рассмотрим основной программный код, связанный с распознаванием действий в отношении узлов и ветвей.

```
//---нажатие левой клавиши мыши в области компонента TPaintBox1---//
void __fastcall TFormMain::PaintBox1MouseDown (TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)                {
newPt=TPoint(X+StartX,Y+StartY); //инициализация точки щелчка мышью
if(X<25||Y<25||X>PaintBox1->Width-25||Y>PaintBox1->Height-25)return;
    switch(nEditType)                          {
    case 1:                              {// режим создания и перемещения узлов
  bool sosed = false;// режим создания узлов, если нет близких к newPt узлов
  for(int i = 0; i<UzelList->Count; i++)                                  {
    TUzel *uz = (TUzel*)UzelList->Items[i];//поиск ближайшего узла к newPt
    if(sqrt((newPt.x-uz->GetX())*(newPt.x-uz->GetX())+(newPt.y-uz->GetY())*\
  (newPt.y-uz->GetY()))<80.0) {CurUz = uz;sosed=true;}                    }
  if(!sosed)  { /*инициализация нового узла и перерисовывание TPaintBox1*/
            }                          };break;
    case 2:                                  // режим создания ненаправленной
    case 3:                              {   // и направленной ветви
  for(int i = 0; i<UzelList->Count; i++)                                  {
    TUzel *uz = (TUzel*)UzelList->Items[i];// поиск узла в окрестности newPt
    if(sqrt((newPt.x-uz->GetX())*(newPt.x-uz->GetX())+(newPt.y-uz->GetY())*\
  (newPt.y-uz>GetY()))<40.0)// если узел найден, то захватить его
      {uz1 = uz;break;}                  }                                  };break;
   case 4:                                   {     //режим удаления узла и
  int ind=0x0FFFFFFF;                              //инцидентных ему ветвей
  for(int i = 0; i<UzelList->Count; i++)                                    {
    TUzel *uz = (TUzel*)UzelList->Items[i];// поиск узла в окрестности newPt
```

```
      if(sqrt((newPt.x-uz->GetX())*(newPt.x-uz->GetX())+(newPt.y-uz->GetY())*\
(newPt.y-uz->GetY()))<40.0)  {uz->Delete();ind = uz->N;}/* удаление узла */  }
      /*сохранение конфигурации сети*/  }              }
//---перемещение курсора в области компонента TPaintBox1---//
void __fastcall TFormMain::PaintBox1MouseMove(TObject *Sender, TShiftState
          Shift, int X, int Y)                                                {
  /* проверка необходимости прокрутки изображения в TPaintBox1 */
  if(CurUz != NULL)       {// если есть захваченный для перемещения узел,
  CurUz->SetX(X+StartX); // происходит его перемещение в текущую точку с
  CurUz->SetY(Y+StartY);}// последующим перерисовыванием TPaintBox1   }
//---отпускание левой клавиши мыши в области компонента TPaintBox1---//
void __fastcall TFormMain::PaintBox1MouseUp(TObject *Sender, TMouseButton
          Button,TShiftState Shift, int X, int Y)                  {
   if(uz1 && (nEditType==2 || nEditType==3) )          {//запоминание в newPt
     TPoint newPt = TPoint(X+StartX,Y+StartY);        //точки отпускания мыши,
     // если в режиме создания ветви идентифицирован узел начала
     TUzel *bUz;          //установление ближайшего к newPt узла
     double rast=0.0;
     for(int i = 0; i<UzelList->Count; i++)                              {
       TUzel *uz = (TUzel*)UzelList->Items[i];// перебор по всем узлам
       double rast1 = sqrt((newPt.x-uz->GetX())*(newPt.x-uz->GetX())+\
      (newPt.y-uz->GetY())*(newPt.y-uz->GetY()));
       if(rast == 0.0 || rast>rast1 || rast<80){rast = rast1;bUz=uz;}          }
   if (uz1->N==bUz->N)return;// проверка того, что узлы начала и конца разные
   bool inList = false;    // проверка наличия создаваемой ветви в списке ветвей
   for(int i = 0; i<VetvList->Count; i++)                              {
       TVetv *vt = (TVetv*)VetvList->Items[i];//перебор по всем ветвям
       if( (vt->GetX(1)== uz1->GetX() && vt->GetY(1)==uz1->GetY() &&
      vt->GetX(2)== bUz->GetX() && vt->GetY(2)==bUz->GetY())||
      (vt->GetX(1)== bUz->GetX() && vt->GetY(1)==bUz->GetY() &&
      vt->GetX(2)== uz1->GetX() && vt->GetY(2)==uz1->GetY()) )
       inList = true;                                                      }
   if(!inList) {/* инициализация ветви, если ее нет в списке ветвей */}
   uz1 = NULL;   /* обнуление узла начала ветви */   }
   /* перерисовывание TPaintBox1 */                              }
```

Как видно из приведенного листинга, при возникновении события *PaintBox1MouseDown* (/*другие параметры*/, int *X*, int *Y*) проводится проверка наличия узла в окрестности точки с координатами *X* и *Y*. Если такой узел существует, то он, в зависимости от значения глобальной переменной nEditType, либо захватывается (для последующего перемещения или инициализации узла начала ветви), либо удаляется. В противном случае происходит инициализация нового узла при nEditType, равном 1. Здесь следует отметить, что переборы узлов осуществляются в соответствии с методом сильнейшего конкурента (МСК), описанным в теории распознавания образов.

При возникновении события *PaintBox1MouseMove* (/*другие параметры*/, int *X*, int *Y*), если есть захваченный узел и значение nEditType равно 1, происходит его перемещение в точку с координатами *X* и *Y*. При этом отслеживается, чтобы при необходимости осуществлялась прокрутка изображения (координаты смещения изображения TPaintBox1 от левого верхнего угла запоминаются в глобальных переменных *StartX* и *StartY*), что позволяет вводить практически неограниченное число элементов графа.

При возникновении события *PaintBox1MouseUp* (/*другие параметры*/, int *X*, int *Y*), если есть захваченный узел в режиме создания ветви, происходит поиск ближайшего узла в окрестности точки с координатами *X* и *Y* . При этом во всех вышеописанных случаях выполняется перерисовывание элементов, содержащихся в TPaintBox1. Здесь представляется важным отметить, что элементы могут быть представлены любым изображением, однако для наглядности следует ограничить их размеры окрестностью, заданной при использовании МСК.

Из приведенного листинга также видно, что при инициализации узлов и ветвей могут быть заданы любые дополнительные параметры, что не повлияет на картину распознавания элементов графа. Для этого достаточно адекватно описать в классах узла и ветви (в данном случае это TUzel и TVetv) соответствующие свойства и методы. Также очевидно, что описанные алгоритмы применимы не только при использовании техники «Drag–And–Drop», но и при любом другом способе ввода и коррекции данных. При этом представляется целесообразным хранить информацию об узлах и ветвях в связанных списках (в данном случае это TUzelList и TVetvList), чтобы избежать работы с сильно разреженными матрицами, т.к. для реальных информационных и технических систем число узлов и ветвей обычно достаточно велико и сопоставимо по порядку величины.

Ввиду того, что при инициализации и изменении узлов и ветвей могут быть представлены другие необходимые характеристики элементов, представленные идеи применимы для визуализации структурных особенностей любой как информационной, так и технической системы. Такие характеристики, адекватно представленные с позиции объектно-ориентированного программирования, позволяют выполнять задачи моделирования и оценивания.

Выводы:

1. Предложено применение объектно-ориентированного подхода программирования для распознавания образов элементов двухполюсных структур на основе использования связанных списков.

2. Рассмотрена программная реализация визуального создания и редактирования элементов графа.